IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 1, March, 2013
**ISSN: 2320 - 8791**
**www.ijreat.org**

# Survey on Mitigation of DOS and DDOS Attacks in the Presence of Clock drift

**C.Kavitha[1], S.Mohana2, Mrs.A.Karmel[3]**

**[1,2]Department of Computer Science and Engineering/ Agni College of Technology/ Final Year, Chennai, Tamil Nadu/India**

**[3]Department of Computer Science and Engineering/ Agni College of Technology/Assistant Professor, Chennai, Tamil Nadu/India**

**Abstract**
Distributed Denial of Service (DDoS) Attacks which can be so powerful that they can easily deplete the computing resources or bandwidth of the potential targets. DDoS attacks can be accosted in two levels: application-level and network-level. The weak point in network-based application is that the communication port is commonly open .This allows attackers to possibly launch Denial of Service (Dos) Attacks. Solutions to this problem are, we use port hopping technique to support many clients without the need of group synchronization in the presence of clock drift. Furthermore we use HOPERAA algorithm and BIGWHEEL algorithm to overcome distributed denial of service attacks.

**Keywords**: *Network Attacks, Port number, Contact initiation, Distributed Denial of Service Attack.*

## 1. Introduction

A Network attack is a threat, intrusion, and denial of service or other attack on a network infrastructure that will analyze your network and gain information to eventually cause your network to crash or to become corrupted.

There are at least seven types of network attacks. They are *mapping, Spoofing attack, Sniffing attack, Hijacking, Trojans, Social engineering, DoS and DDoS*. But this paper describes about Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) attacks.

Denial of Service (DoS) attack is an attempt to make machine or network resource unavailable to its intended users by disrupting it, crashing it, jamming it or flooding. The motivation for DoS attacks is not to break into a system. One can say that this will typically happen through following means:

1. Crashing the system. Deny communication between systems.
2. Bring the network or the system down or have it operate at a reduced speed which affects productivity.
3. Hang the system, which is more dangerous than crashing since there is no automatic reboot.

DoS attacks can also be major components of other type of attacks. There are many types of DoS attacks. Among them important attacks that exist are Teardrop attack, Bandwidth attack, Blind attack, SYN flood attack and Smurf attack.

1) Teardrop attack sends incorrect IP fragments to the target server.So the server gets crashed if it does not implement TCP/IP fragmentation reassembly code properly.

2) A bandwidth attack is where an attacker tries to consume the available bandwidth of a network by sending a flood of packets. They attacks server fast in a kbps speed.

3) With a blind attack the attacker uses one or more forged IP addresses, which is extremely difficult for the server to filter those packets.

4) A SYN flood is a form of denial of service attack in which an attacker sends a succession of SYN requests to a target's system in an attempt to consume enough server resources to make the system unresponsive to legitimate traffic.

5) In smurf attack, the attacker sends an IP ping request to any website. The ping packet is broadcast to a number of hosts within that site's local network.

The request is from another site and it is the target site that receives the denial of service attack.

A Distributed Denial of Service (DDoS) attack is the combined effort of several machines to bring down victim. It occurs when multiple compromised systems or multiple attackers flood the bandwidth or resources of a targeted system with useless traffic.

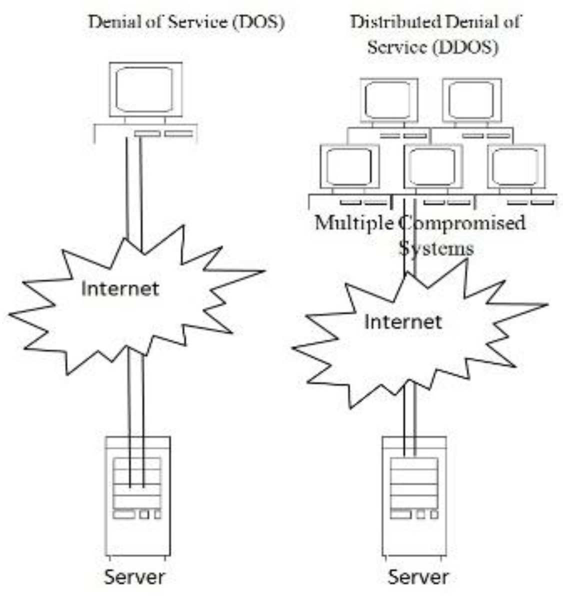

Denial of Service (DOS)
Distributed Denial of Service (DDOS)
Multiple Compromised Systems
Internet
Internet
Server
Server

Fig 1: Denial of Service attack and Distributed Denial of Service attack

Most of the time, attackers collect many (millions) of *zombie machines or bots* .In many cases there is a master machine that launches the attack to zombie machines that are part of a bot network. Some bot networks contain many thousands of machines used to launch an attack

To avoid all those attacks we use port –hopping technique in the presence of *clock drifts*, which *implies* clock values, can vary arbitrarily much with time and in multiparty applications. The application parties communicate via ports that change periodically over time using the pseudorandom function. This method was inspired from the well known frequency hopping paradigm used in signal communication protocols [2]. The focus in that area is to find hopping sequences with the optimal Hamming Correlation Properties [6],[7],[8] But in the earlier solutions, port-hopping support between pairs of processes which are synchronous or exchange acknowledgements. Acknowledgment, if lost, can cause a port to be open for longer time .This become targets to DoS attack themselves.
We propose two algorithms in proposed to avoid attacks. One is HOPERAA (Hopping-Period-Align-and-Adjust) algorithm, executed by each client to adjust its period length and align its hopping time with the server. Second is BIGWHEEL algorithm enable multiparty communication with port hopping, this algorithm for a server to support hopping with many clients.

The basic idea in both algorithms is that each client interacts independently with the server and considers the server's clock as the point of reference clock. In this algorithm, there is no need for group synchronization which would raise scalability issues. The HOPERAA and BIGWHEEL algorithm's detailed explanations are in the Section 3.2.3 and 3.3 respectively.

## 1.1 Problem analysis

Problem that an adversary wants to subvert the communication of client-server application is by attacking the communication channels. At each time point, some ports must be open at the server side to receive the messages sent from legitimate clients The N which denotes the size of port number space, meaning that there are N ports that the server can use for communication at the server side.
The server and the legitimate clients share a pseudorandom function .It generate the port numbers which will be used in the communication. We assume there exists a preceding authentication procedure. It enables the server to distinguish the messages from the genuine clients. We assume that every client is honest which means any execution of the client is based on the protocol and clients will not reveal the random function to the adversary. The attacker is modeled as an adaptive adversary who can eavesdrop and attack a bounded number of ports simultaneously.

### 1.2 System architecture

Clock drift is used to maintain clock rates between client and server. Clock rates between the client and server has been adjusted and aligned to same. Based on the application the client use, Pseudo random function generates pseudo random seed in the server and it assigns ports to each client.

*Contact messages*: (1) Client to send message to a port that is already closed or is not opened yet and then client align the hopping time period at adversary chosen time intervals to control the align. (2) HOPERRA Executed by each client to adjust its hopping period length and align its hopping period with the server.

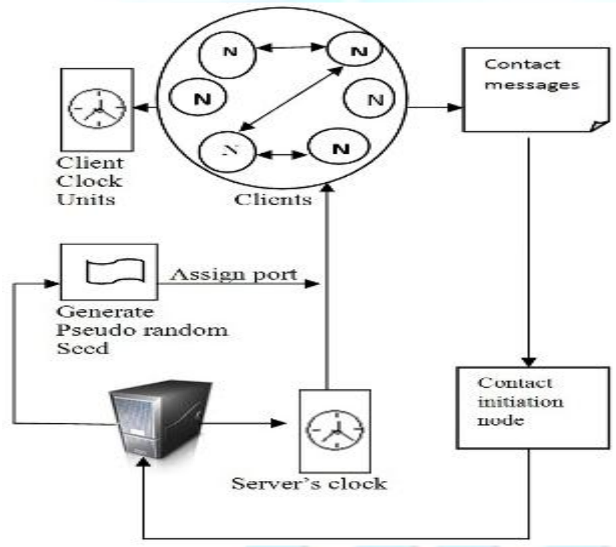


Fig 3: System Architecture

## 2. Related work

Author Badishi et al. [3] consider the problem of overcoming (Distributed) Denial of Service (DoS) attacks by realistic adversaries that have knowledge of their attack's successfulness; His solution for this problem in a high-speed network environment necessitates lightweight mechanisms for differentiating between valid traffic and the attacker's packets. He says that the main challenge in presenting such a solution is to exploit existing packet filtering mechanisms. It is in a way which allows fast processing of packets, that is complex enough so that the attacker cannot efficiently craft packets that pass the filters. He show a protocol that mitigates DoS attacks by adversaries that can eavesdrop and (with some delay) adapt their attacks accordingly and he show that his protocol provides effective DoS prevention for realistic attack and deployment scenarios.

Author Lee Thing [4] propose a another technique, called port hopping where the UDP/TCP port number used by the server varies as a function of time and a shared secret between the server and the client. The main strength of the mechanism lie in the simplification of both the detection and filtering of malicious attacks packets. This port hopping technique is compatible with the UDP and TCP protocols which can be implemented using the socket communications for the UDP protocol, and for setting up TCP communications. His experiments show that the port hopping technique is effective in detecting and filter out the malicious traffic.

Author Srivatsa et al. [5] propose a light weight client-transparent technique to defend against DoS attacks with two unique features: (i) his technique can be implemented entirely using JavaScript support provided by a standard client-side browser like Mozilla Firefox. Client transparency which follows from the fact that: (i) no changes to client-side software are required (ii) no client-side super user privileges are required, and (iii) clients (human beings or automated clients) can browse a DoS protected website in the same manner that they browse other websites (ii) Although he operate using the client-side browser (HTTP layer), His technique enables fast IP level packet filtering at the server's firewall and requires no changes to the application(s) hosted by the web server. In his paper he presents a detailed design of his technique along with a detailed security analysis. He also describes a concrete implementation of our proposal on the Linux Kernel and presents an evaluation using two applications: bandwidth intensive Apache HTTPD and database incurs a low performance overhead and is resilient to DoS attacks.

## 3. Solutions to the Problem

### 3.1 Pseudorandom function

Clients get the seed from Server for the pseudorandom function to compute the port sequence. The application data is sent from Client to Server which is sent out to the open ports of Server that changes every time units of Server clock, corresponding to client time units in Client's clock. This happens after the contact-initiation part. Periodically, the sender and receiver can use new seeds of the pseudorandom function to generate different port number sequences. This allows the port number sequence which is used for communication is changed periodically. The clients and the server share a pseudorandom function to compute which port should be used in a certain time slot. Every client uses the same pseudorandom function to generate the destination port number.

## 3.2 Port Hopping

Port hopping mechanism consists of three parts: the *contact initiation part, the data transmission part, and the resynchronization/adjustment part* which is controlled by the *Hopping Period Alignment and Adjustment (HOPERAA) algorithm*. There are basically two ports called worker ports and guard ports.

The ports which are open in the server side for receiving the data messages from the client are called *worker ports*. The ports which are open in the server side for receiving the contact initiation messages from the client called *guard ports.*

### *3.2.1 Contact-Initiation Part*

To enable Client C to initiate contact with S without having Server listen at a "well-known" port and without relying on a third party, the timestamp of the corresponding contact-initiation message received by the server, and t2 is the arriving time of the same message. They will be stored later by the client for estimating its clock drift. The server divides the range of port numbers into k intervals evenly and opens k different guard ports at the same time, keeping one guard port per one interval. It changes them for every Length of the server's hopping period time units but in each interval, it still keeps one open guard port.
Client sends contact-initiation messages to all the ports in an interval which is randomly chosen. When Server receives a contact-initiation message, it replies with the seed for the pseudorandom function and it will send the index for computing the next worker port to the clients. When the next worker port is open, the server will send that reply message which will happen in Length of the server's hopping period time units, because the network may lose messages and open ports can be disabled by the adversary, Client may not get the reply from Server. To save bandwidth, instead of keeping on sending contact-initiation messages, Client will set a timeout for waiting the reply message.
The timeout is set to time units, taking into account the message round trip time and the waiting time by the server to send the reply .Until it reaches the timeout., if Client does not receive a reply, then it will choose another interval of port numbers and send contact-initiation messages again until it gets the reply.

**Algorithm 1.Algorithm for Client C in the initiation stage**

```
T_c ← undef
Reply←false
-Send contact initiation messages phase:
while reply=false do
I← select ($I_i$|i ∈{1, 2, .k})
    for all p∈ I do
        send (init, time, p)
    end for
    wait (2µ+L)
end while

-receive reply
phase:
receive(reply, σ,h1,t1)
if reply=false then
reply= true
$T_c$=0
Start sending date
end if
```

**Algorithm 2.Algorithm for Server S in the initiation stage**

```
– receive contact-initiation message phase:
receive (init,time,p)
   $t_1$ ← $Time_{now}$
   if $session_C$ = undef  then
  open (session,C)
```

```
        h←time
        end if
    wait until next worker port $p_i$ opens send(reply, σ,timestamp,$h_1$,$t_1$)
```

### *3.2.2 Data transmission part*

In the data transmission stage, the hopping time of Client will drift apart from the server's. This might cause Client to send messages to a port that is already closed or is not opened yet, depending on whether Client's clock is slower or faster than Server clock. Deviation of hopping times would imply more message loss, so Client has to align the hopping time at adaptively chosen time intervals to control the align called the HOPERAA execution-intervals to keep the offset of the open times counted by the server and the client of a worker port within time units, then the HOPERAA execution interval. The client has no idea about its clock drift. Every contact-initiation message and reply message will be attached with the timestamp of its sending time. The reply message also has the timestamp and the arrival time of the first contact-initiation message received by the server.

**Algorithm 3. Algorithm for C in data transmission stage**

```
Seq ← 0
$P_{old}$ ← f (σ)
$P_{new}$← f ((σ+ 1))
-$S_A$ (*sending the messages*)
    while true do
        send (Data,$P_{old}$)
        if $(i - 1) L \leq T_c \leq iL - \mu$ then
            Send (Data,$P_{new}$)
            end if
    end while
    – $U_A$ (*changing the destination port*)
    {$T_c$=iL}
    $P_{old}$ ← $P_{new}$
    $P_{new}$ ← f ((σ+ i + 1))
```

Client sends the data messages to the worker ports of server. After server sends the reply in the contact –initiation part .Client gets the seed of the pseudorandom function f which generates the sequence of the worker ports. L+μ time units are taken as the open interval in the worker ports. μ Time units are taken as the new worker port which will be opened earlier than the waiting time of the old one.

Client sends the contact initiation message received by server which sends the reply message at the time when next worker port is opened. And σ denotes the integer for pseudorandom function to generate port number. When server replies with a integer, client will send data messages to the port computed from f.When $T_c$=iL, the destination port number of the data messages will be computed.

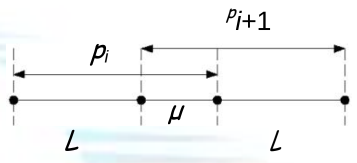


Fig 4: Worker port's open interval with overlap

### *3.2.3 Resynchronization/Adjustment part*

HOPERAA algorithm, which is an *adaptive algorithm*, enables hopping in the presence of clock drift. Client clock has a drift related to server clock, if they have different clock rates, then length of the hopping period will deviate from each other. This causes message loss which is due to the fact that client may send message to some of server's port which have been closed or have not been opened yet-depending on client's clock runs slower or faster than server's clock respectively.

To solve we propose HOPERAA algorithm by each client to adjust its hopping period length and align its hopping time with the server. Growth of deviation of hopping times would imply more message loss, so client has to align the hopping time at adaptively chosen time intervals which are called the HOPERAA execution-intervals. If the client's clock rate is slower than the server's clock rate, which means clock drift of client is less than 1, and if we want to keep the offset of the closing times counted by the server and the client of a worker port within maximum allowed value time units. However, the client has no idea about its clock drift.

We suggest a method which exchange messages with information about the sending and receiving times (time stamped with local clock values) between client and server, to estimate the clock drift.

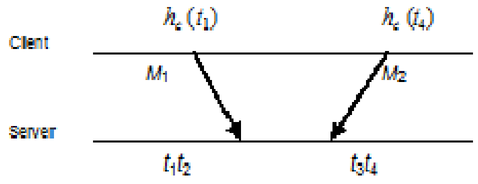


Fig 5: Messages exchange and associated times and Timestamps.

## 3.3 BIGWHEEL Algorithm

BIGWHEEL algorithm enables multiparty applications which mean many clients per server. Clients can interact with the server independently. For scalability reasons, it is desirable that the server has more than one worker ports open in each time period, so as to balance the load among them. Having the same hopping period and different phases in the corresponding hopping sequences ,such a method can manage to bound better the time it takes for each client to initiate contact with the server. Consider an example *Big Wheel rides at amusement parks*: the next available compartment is queued by the clients where each compartments are represented as a hopping sequences; compartments are deployed in a way that aims at balancing the load among them .It also minimizes the waiting time of the client to initiate contact with the server.

## 4.Simulation and Results

The simulation was done using NS-2 simulator to evaluate the performance of our DDoS detection with results obtained from the experiment. We tested in fedora environment. This section introduces the experimental setup and reports performance results.

1)Experiment study
Our simulation includes 12 clients, 7 intermediate routers and 3 servers as shown in figure . The bandwidth of legitimate traffic is set constant and the simulation of attack traffic is achieved by randomly generating many pairs of Constant Bit Rate (CBR) TCP flows in NS2.

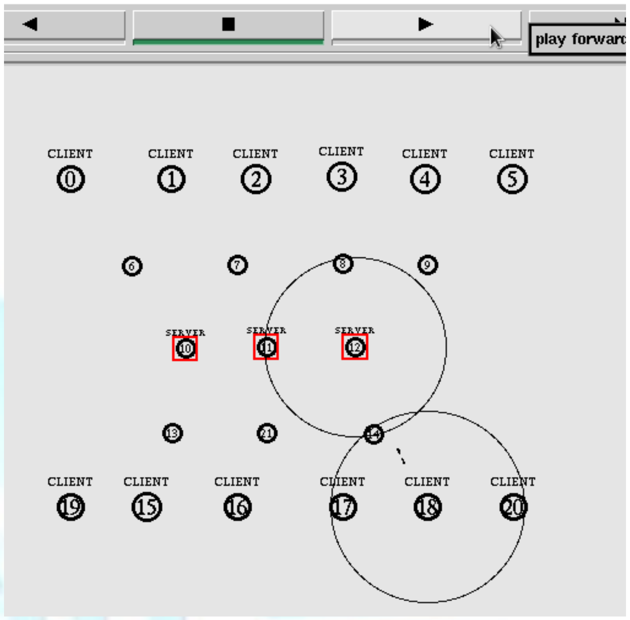


Fig 6:Client send request to server

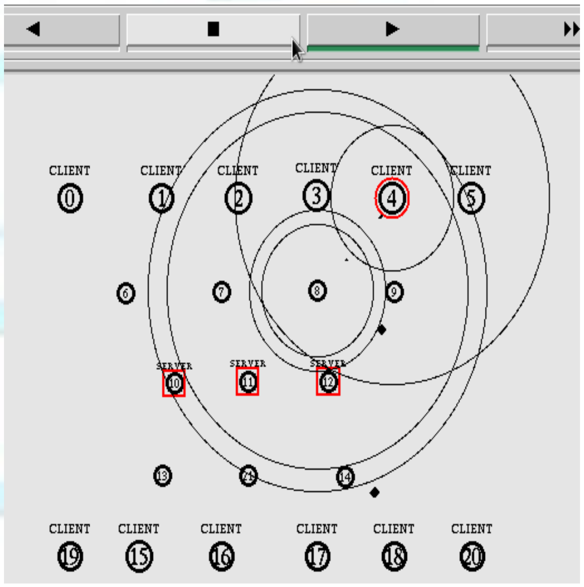
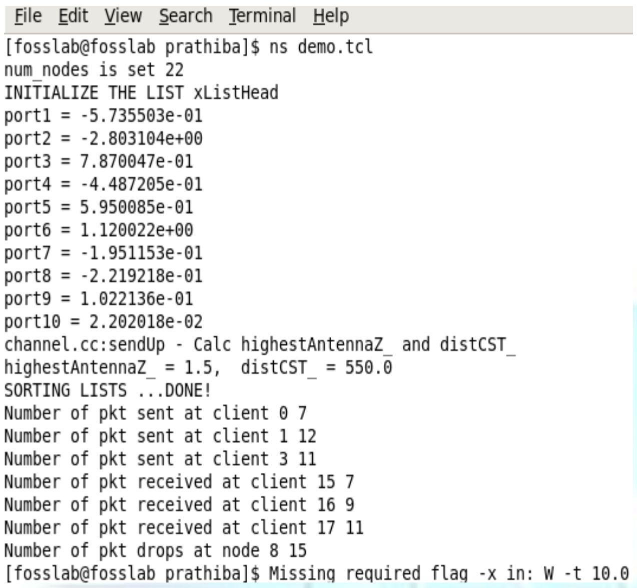


Fig 7:Attack identified and message loss



```
[fosslab@fosslab prathiba]$ ns demo.tcl
num_nodes is set 22
INITIALIZE THE LIST xListHead
port1 = -5.735503e-01
port2 = -2.803104e+00
port3 = 7.870047e-01
port4 = -4.487205e-01
port5 = 5.950085e-01
port6 = 1.120022e+00
port7 = -1.951153e-01
port8 = -2.219218e-01
port9 = 1.022136e-01
port10 = 2.202018e-02
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
Number of pkt sent at client 0 7
Number of pkt sent at client 1 12
Number of pkt sent at client 3 11
Number of pkt received at client 15 7
Number of pkt received at client 16 9
Number of pkt received at client 17 11
Number of pkt drops at node 8 15
[fosslab@fosslab prathiba]$ Missing required flag -x in: W -t 10.0
```

Fig 8:Port number initialization

**2)** Performance Evaluation:

To evaluate the performance of our algorithm, we plot the evaluation graph which contains time value in X-axis and normalize entropy value in Y-axis. With the help of graph shown below, we are easily able to conclude that if we take threshold value1.1, it can easily detect the attack.

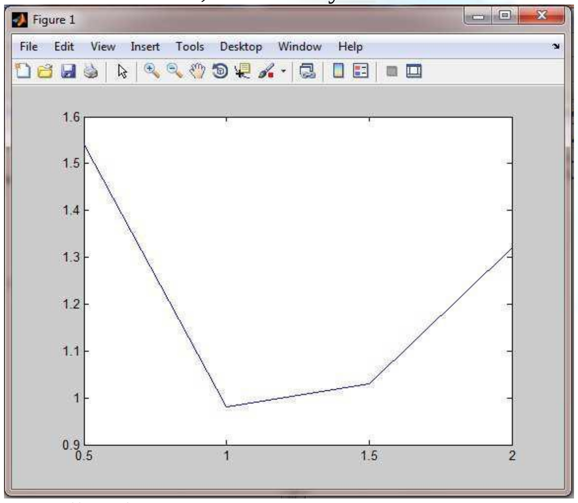


Fig.8 Effect of DDOS attack

## 4. Experimental Results

1) DoS and DDoS attacks can be made less severe by performing these algorithms.
2) The message loss due to the clock drifts can be controlled by adjusting parameters in the protocol.
3) The Malicious traffic and congestion in the network has been reduced and had improved the lifetime of the network and the latency.
4) Server can handle packets from the legitimate user effectively and delay may not occur.
5) Server's performance may not get slow down.
6) There will be no attempt to consume a server's resources (network bandwidth, computing power, main memory, disk bandwidth etc) to near exhaustion. So that there will be resources left to handle requests from legitimate clients.
7) Utilization of multiple machines (often thousands) by attackers to generate excessive traffic can be avoided.

## 5. Conclusions

In this paper we have presented a detailed study on how to mitigate DoS and DDoS attacks in the presence of clock drift. In this work, we investigate application-level protection against DoS attacks. An algorithm is presented for a server to support port hopping with many clients. A main conclusion is that it is possible to employ the port hopping method in multiparty applications in a scalable way. Another main conclusion is that the adaptive method can work under timing uncertainty and specifically fixed clock drifts.
In the Future work, we can extend it to prevent Network attacks. There are many types of network attack, we investigate to prevent server from all such types of network attacks.

## Acknowledgements

This paper has benefited from conversations with many different people – far more than can be acknowledged completely here. Still we would like to particularly thank Dr.P.S.K.PATRA, HOD, CSE for his guidance and support.

## References

[1]Z. Fu, M. Papatriantafilou, and P. Tsigas, "Mitigating Distributed Denial of Service Attacks in Multiparty Applications in the Presence of Clock Drifts," IEEE Transaction on dependable and secure computing, vol 9, no 3, May/June 2012.

[2]Spread Spectrum Scene, http://sss-mag.com/ss.html, 2011.

[3] G. Badishi, A. Herzberg, and I. Keidar, "Keeping Denial-of-Service Attackers in the Dark," IEEE Trans. Dependable and Secure Computing, vol. 4, no. 3, pp. 191-204, July-Sept. 2007.Greece, and the PhD degree in computer engineering and informatics from the same University, in 1994.

[4]H. Lee and V. Thing, "Port Hopping for Resilient Networks," Proc.IEEE 60th Vehicular Technology Conf. (VTC2004-Fall), vol. 5, pp. 3291-3295, 2004

[5]M. Srivatsa, A. Iyengar, J. Yin, and L. Liu, "A Client-Transparent Approach to Defend against Denial of Service Attacks," Proc. IEEE 25th Symposium Reliable Distributed Systems (SRDS '06), pp. 61-70, 2006.

[6] A. Lempel and H. Greenberger, "Families of Sequences with Optimal Hamming Correlation Properties," IEEE Trans. Information Theory, vol. IT-20, no. 1, pp. 90-94, Jan. 1974.

[7]G. Ge, R. Fuji-Hara, and Y. Miao, "Further Combinatorial Constructions for Optimal Frequency-Hopping Sequences," J. Combinatorial Theory Series A, vol. 113, no. 8, pp. 1699-1718, 2006.

[8]Y.M. Ryoh Fuji-Hara and M. Mishima, "Optimal Frequency Hopping Sequences: A Combinatorial Approach," IEEE Trans.Information Theory, vol. 50, no.10, pp.2408-2420, Oct. 2004.